On-Device Federated Learning via Blockchain and its Latency Analysis

Preprint · August 2018

CITATIONS
0

READS
1,244

4 authors, including:

Jihong Park
University of Oulu
61 PUBLICATIONS **465** CITATIONS
SEE PROFILE

Mehdi Bennis
University of Oulu
377 PUBLICATIONS **8,728** CITATIONS
SEE PROFILE

On-Device Federated Learning via Blockchain and its Latency Analysis

Hyesung Kim, Jihong Park†, Mehdi Bennis†, and Seong-Lyun Kim

***Abstract*—In this letter, we propose a block-chained federated learning (BlockFL) architecture, where mobile devices' local learning model updates are exchanged and verified by leveraging blockchain. This enables on-device machine learning without any central coordination, even when each device lacks its own training data samples. We investigate the end-to-end learning completion latency of BlockFL, thereby yielding the optimal block generation rate as well as important insights in terms of network scalability and robustness.**

***Index Terms*—On-device machine learning, federated learning, distributed ledger technology, blockchain, latency.**

I. Introduction

Future wireless systems are envisaged to ensure low latency and high reliability anywhere and anytime [1]–[3]. Even when a mobile device loses connectivity, the device needs to make decisions via a high-quality machine learning model. Training such an on-device learning model commonly requires a much larger number of data samples compared to the samples available at each device, and necessitates data sample exchanges with other devices [4], [5]. In this letter, we consider the problem of training each device's local model by exchanging the local data samples without any central coordination.

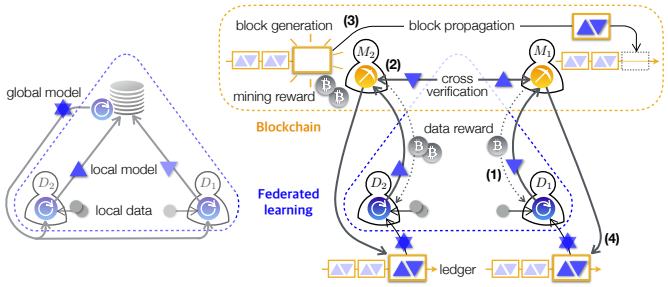
One key challenge is that local data samples are owned by each device. Thus, the exchanges should keep the raw data samples private from other devices, e.g., for patient data. To this end, as proposed in Google's federated learning (FL) [4], [5], hereafter referred to as a vanilla FL, each device can exchange its *local model update*, i.e., learning model's weight and gradient parameters, which is more privacy-preserving compared to sharing the raw data samples. As illustrated in Fig. 1-a, the vanilla FL's exchange is enabled by the aid of a central server that aggregates all the local model updates and takes an ensemble average, yielding a *global model update*. Then, each device downloads the global model update, and computes its next local update until the global model training is completed, for instance via a distributed stochastic gradient descent (SGD) approach [5], [6]. Unfortunately, such a centralized operation is vulnerable to the server's malfunction. The resultant inaccurate global model updates distort all local model updates, and the overall training may thereby collapse, calling for a distributed FL architecture.

Another important challenge brought by the aforementioned local data ownership is the reward system for the local devices. In fact, a device having a larger number of data samples contributes more to the global model training, while consuming more computation power and/or time. On the one hand, without providing proper compensation proportional to the number of samples, such a device is less willing to federate with the other devices possessing few data samples. On the other hand, as a side effect of the compensation, some untruthful devices may pretend to have a larger number of data samples than their actual sample sizes, yielding inaccurate global model updates in FL.

In order to resolve the issues of private exchange and reward mechanism, by leveraging *blockchain* [7], [8] instead of a central entity, we propose a *block-chained FL (BlockFL)* architecture, where the blockchain network enables exchanging the devices' local model updates while verifying and providing their corresponding rewards. As shown in Fig. 1-b, the logical structure of BlockFL consists of mobile devices and miners. The miners can physically be either randomly selected devices or separate nodes like a conventional blockchain network [7]. The operation of BlockFL is summarized as follows.

H. Kim and S.-L. Kim are with the Radio Resource Management & Optimization Laboratory, Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea (email: {hskim, slkim}@ramo.yonsei.ac.kr).

†J. Park and †M. Bennis are with the Centre for Wireless Communications, University of Oulu, 4500 Oulu, Finland (email: {jihong.park, mehdi.bennis}@oulu.fi).



(a) Vanilla FL. (b) Proposed BlockFL.

Fig. 1. An illustration of (a) the vanilla federated learning (FL) [4], [5] and (b) the proposed block-chained FL (BlockFL) architectures.

(1) Each device in BlockFL computes and uploads the local model update to its associated miner in the blockchain network, while in return receiving the data reward proportional to the number of its data samples from the miner.
(2) Miners exchange and verify all the local model updates, and then run the Proof-of-Work (PoW) [7].
(3) Once a miner completes the PoW, it generates a block where the verified local model updates are recorded, and receives the mining reward from the blockchain network.
(4) Finally, the generated block storing the aggregate local model updates is added to a blockchain, also known as distributed ledger, and is downloaded by the devices. Each device computes the global model update from the freshest block, which is an input of the next local model update.

It is worth noting that the global model update of BlockFL is computed locally at each device. Therefore, a miner's and/or a device's malfunction during the global model update process does not affect other devices' local global model updates, ensuring the robustness of the overall training.

For the sake of these benefits, in contrast to the vanilla FL, BlockFL needs to pay for the extra delay incurred by the blockchain network. To address this, the end-to-end latency model of BlockFL is formulated by taking into account communication, computation, and the PoW delays during the FL and blockchain operations. The resulting latency is minimized by adjusting the block generation rate, i.e., the PoW difficulty.

II. Architecture and Operation

This section describes the individual operation of FL and blockchain in BlockFL, followed by their joint operation in detail. We use the subscripts i and j that identify different devices and miners, respectively. The subscript k describes different data samples. The superscript ℓ distinguishes the global model update iterations, referred to as epochs.

A. FL operation in BlockFL

The FL under study is operated by a set of devices $\mathcal{D} = \{1, 2, \cdots, N_D\}$ with $|\mathcal{D}| = N_D$. The i-th device D_i owns a set of data samples $\mathcal{S}_i$ with $|\mathcal{S}_i| = N_i$, and trains its local model. The local model updates of the device D_i is uploaded to its associated miner M_i that is uniformly randomly selected out of a set of miners $\mathcal{M} = \{1, 2, \cdots, N_M\}$, where $M_j \in \mathcal{M}$ with $|\mathcal{M}| = N_M$. $\mathcal{M} = \mathcal{D}$ is satisfied if the miners are physically

identical to the devices, otherwise we have $\mathcal{M} \neq \mathcal{S}$. Next, the total number N_D of the local model updates are verified and exchanged through the miners, and finally the aggregate local model updates are downloaded from each miner to its associated device.

For the sake of convenience, our distributed model training focuses on solving a linear regression problem in a parallel manner. The regression considers a set of the entire devices' data samples $\mathcal{S} = \cup_{i=1}^{N_D} \mathcal{S}_i$ with $|\mathcal{S}| = N_S$. The k-th data sample $s_k \in \mathcal{S}$ is given as $s_k = \{x_k, y_k\}$ for a d-dimensional column vector $x_k \in \mathbb{R}^d$ and a scalar value $y_k \in \mathbb{R}$. The regression objective is minimizing a loss function $f(w)$ with respect to a d-dimensional column vector $w \in \mathbb{R}^d$, denoted as a global weight. For simplicity, the loss function $f(w)$ is chosen as the mean squared error (MSE):

$$f(w) = \frac{1}{N_S} \sum_{i=1}^{N_D} \sum_{s_k \in \mathcal{S}_i} f_k(w), \tag{1}$$

where $f_k(w) = (x_k^\top w - y_k)^2/2$ and the notation $(\cdot)^\top$ indicates the vector transpose operation. Different loss functions under more complicated neural network models can readily be incorporated with minor modifications, as done in [9], [10].

In order to solve the aforementioned regression problem, following the vanilla FL settings in [4], the learning model of the device D_i is locally trained with the set $\mathcal{S}_i$ of its data samples via a stochastic variance reduced gradient (SVRG) algorithm [11], and all devices' local model updates are aggregated using a distributed approximate Newton (DANE) method [12], yielding the global model update.

To elaborate, the global model is updated up to L epochs. For each epoch, the device D_i's local model is updated with the number N_i of iterations. At the t-th local iteration of the ℓ-th epoch, the local weight $w_i^{(t,\ell)} \in \mathbb{R}^d$ of the device D_i is:

$$w_i^{(t,\ell)} = w_i^{(t-1,\ell)} - \frac{\beta}{N_i} \left(\left[\nabla f_k(w_i^{(t-1,\ell)}) - \nabla f_k(w^{(\ell)}) \right] + \nabla f(w^{(\ell)}) \right), \tag{2}$$

where $\beta > 0$ is a step size, $w^{(\ell)}$ indicates the global weight at the ℓ-th epoch, and $\nabla f(w^{(\ell)}) = 1/N_S \cdot \sum_{i=1}^{N_D} \sum_{s_k \in \mathcal{S}_i} \nabla f_k(w^{(\ell)})$ is obtained from (1). Let $w_i^{(\ell)}$ denote the local weight after the last local iteration of the ℓ-th epoch, i.e., $w_i^{(\ell)} = w_i^{(N_i,\ell)}$. Then, the global weight $w^{(\ell)}$ is updated as:

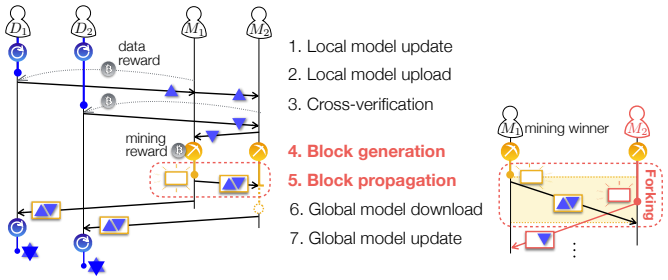
$$w^{(\ell)} = w^{(\ell-1)} + \sum_{i=1}^{N_D} \frac{N_i}{N_S} \left(w_i^{(\ell)} - w^{(\ell-1)} \right). \tag{3}$$

These local and global weight updates continue until the global weight $w^{(L)}$ satisfies $|w^{(L)} - w^{(L-1)}| \leq \varepsilon$ for a constant $\varepsilon > 0$.

In the vanilla FL structure in [4], [5], at the ℓ-th epoch, the device D_i uploads its local model update $(w_i^{(\ell)}, \{\nabla f_k(w^{(\ell)})\}_{s_k \in \mathcal{S}_i})$ to the central server, with the model update size δ_m that is identically given for all devices. The global model update $(w^{(\ell)}, \nabla f(w^{(\ell)}))$ with the same size δ_m is computed by the server, which is downloaded to all devices. In BlockFL, the server entity is substituted with a blockchain network as detailed in the following subsection.

B. Blockchain operation in BlockFL

In the blockchain network of BlockFL, the blocks and their verification by the miners in $\mathcal{M}$ are designed so as to exchange the local model updates truthfully through a distributed ledger. Each block in a ledger is divided into its body and header parts. In the conventional blockchain structure [7], the body contains a list of verified transactions. In BlockFL, the body stores the local model updates of the devices in $\mathcal{D}$, i.e., $(w_i^{(\ell)}, \{\nabla f_k(w^{(\ell)})\}_{s_k \in \mathcal{S}_i})$ for the device D_i at the ℓ-th epoch, as well as its local computation time $T_{\mathrm{local},i}^{(\ell)}$ that is discussed at the end of this subsection. Following the structure in [7], the



(a) Without forking. (b) With forking.

Fig. 2. The one-epoch operation of BlockFL with and without forking.

header contains the information of a pointer to the previous block, block generation rate λ, and the output value of the PoW, referred to as nonce. In order to store all devices' local model updates, the size of each block is set as $h+\delta_\mathrm{m}N_D$, where h and δ_m are the header and model update sizes, respectively.

Each miner has a candidate block that is filled with the local model updates from its associated devices and/or other miners, in the order of arrival. The filling procedure may continue until it reaches the block size or a maximum waiting time T_wait measured from the beginning of each epoch. For simplicity, we assume T_wait is sufficiently long such that each block is always filled with all devices' local model updates.

Afterwards, following the PoW [7], the miner keeps generating a random number until the number, i.e., nonce, becomes smaller than a target value. Once the miner M_1 succeeds in finding the nonce, its candidate block is allowed to be generated as a new block as shown in Fig 2. Here, the block generation rate λ can be controlled by adjusting the PoW difficulty, e.g., the lower PoW target value, the smaller λ.

Next, the generated block is propagated to all other miners, in order to synchronize all their distributed ledgers. To this end, as done in [7], all the miners receiving the generated block are forced to stop their PoW operations and to add the generated block to their local ledgers. As illustrated in Fig 2, if another miner M_2 succeeds in its block generation within the propagation delay of the firstly generated block, then some miners may mistakenly add this secondly generated block to their local ledgers, known as *forking*. In BlockFL, forking makes some devices apply an incorrect global model update to their next local model updates. Forking frequency increases with λ and the block propagation delay, and its mitigation incurs an extra delay, to be elaborated in Section III.

In addition to the aforementioned operation for local model update exchanges, as described in Figs. 1 and 2, the blockchain network provides rewards for data samples to the devices and for the verification process to the miners, referred to as *data reward* and *mining reward*, respectively. The data reward of the device D_i is received from its associated miner, and the amount is proportional to the data sample size N_i. When the miner M_j generates a block, its mining reward is earned by the blockchain network, as done in the conventional blockchain structure [7]. The amount of mining reward is proportional to the aggregate data sample size of its all associating devices, namely, $\sum_{i=1}^{N_{M_j}} N_i$ where N_{M_j} denotes the number of devices associated with the miner M_j. This motivates miners to collect more local model updates, while compensating their expenditure for the data reward.

As a side effect of the reward system, some untruthful devices may deceive the miners by inflating their actual sample sizes used for the local model dates or by generating arbitrary local model updates without conducting local learning computation. Miners verify truthful local updates before storing the local model updates in their candidate blocks. The verification is performed by comparing the sample size N_i

with its corresponding computation time $T^{(\ell)}_{\text{local},i}$ that is assumed to be truthful, following the proof of elapsed time [13] under e.g., Intel's SGX technology [14].

C. One-epoch BlockFL operation

As depicted in Fig. 2, the BlockFL operation of the device D_i at the ℓ-th epoch is described by the following seven steps.

0. Initialization (for $\ell = 1$): Initial parameters are uniformly randomly chosen from predefined ranges: the local and global weights $w_i^{(0)}$, $w^{(0)} \in (0, w_{\max}]$ for a constant $w_{\max}$, and the the global gradient $\nabla f(w^{(0)}) \in (0, 1]$.
1. Local model update: The device D_i computes (2) with the number N_i of iterations.
2. Local model upload: The device D_i uniformly randomly associates with the miner M_i; if $\mathcal{M} = \mathcal{D}$, then M_i is selected from $\mathcal{M}\backslash D_i$. The device uploads the local model updates $(w_i^{(\ell)}, \{\nabla f_k(w^{(\ell)})\}_{s_k \in \mathcal{S}_i})$ and the corresponding local computation time $T^{(\ell)}_{\text{local},i}$ to the associated miner.
3. Cross-verification: Miners broadcast the local model updates obtained from their associated devices. At the same time, the miners verify the received local model updates from their associated devices or the other miners in the order of arrival. The truthfulness of the local model updates are validated, if the local computation time $T^{(\ell)}_{\text{local},i}$ is proportional to the data sample size N_i. The verified local model updates are recorded in the miner's candidate block, until its reaching the block size $(h + \delta_{\text{m}} N_D)$ or the maximum waiting time T_{wait}.
4. Block generation: Each miner starts running the PoW until either it finds the nonce or it receives a generated block from another miner.
5. Block propagation: Denoting as $M_{\hat{o}} \in \mathcal{M}$ the miner who first finds the nonce. Its candidate block is generated as a new block that is broadcasted to all miners. In order to avoid forking, an acknowledgement (ACK) signal is transmitted when each miner detects no forking event. Each miner waits until receiving all miners' ACK signals; otherwise, the operation restarts from Step 1.
6. Global model download: The device D_i downloads the generated block from its associated miner.
7. Global model update: The device D_i locally computes the global model update in (3) by using the aggregate local model updates stored in the generated block.

The said one-epoch procedure continues until the global weight $w^{(L)}$ satisfies $|w^{(L)} - w^{(L-1)}| \leq \varepsilon$. Recall that the centralized FL structure [4], [5] is vulnerable to the server's malfunction that distorts all devices' global model updates. Compared to this, each device in BlockFL locally computes its global model update, which is thus more robust to the malfunction of the miners that replace the server entity.

III. End-to-End Latency Analysis

In this section, we consider a reference device $D_o \in \mathcal{D}$ that is randomly selected. Our objective is to derive the optimal block generation rate λ^* minimizing learning completion latency T_o, defined as the total elapsed time during L epochs at the device D_o. This latency is linearly proportional to the ℓ-th epoch latency, i.e., $T_o = \sum_{\ell=1}^{L} T_o^{(\ell)}$. Thus, we hereafter focus only on $T_o^{(\ell)}$ without loss of generality.

A. One-epoch BlockFL latency model

Following the BlockFL operation in Sect. II-C, the device D_o's ℓ-th epoch latency $T_o^{(\ell)}$ is determined by computation, communication, and block generation delays, as detailed next.

First, computation delays are brought by Steps 1 and 7 in Sect. II-C. Let δ_{d} denote a single data sample's size that is given identically for all data samples. Processing δ_{d} with the clock speed f_c requires δ_{d}/f_c. Local model updating delay $T^{(\ell)}_{\text{local},o}$ in Step 1 is thus given as $T^{(\ell)}_{\text{local},o} = \delta_{\text{d}} N_o / f_c$. Likewise, global model updating delay $T^{(\ell)}_{\text{global},o}$ in Step 7 is evaluated as $T^{(\ell)}_{\text{global},o} = \delta_{\text{m}} N_D / f_c$.

Second, communication delays are entailed by Steps 2 and 6 between devices and miners. Measuring the achievable rate by Shannon capacity under additive white Gaussian noise (AWGN) channels, local model uploading delay $T^{(\ell)}_{\text{up},o}$ in Step 2 is computed as $T^{(\ell)}_{\text{up},o} = \delta_{\text{m}} / [W_{\text{up}} \log_2(1 + \gamma_{\text{up},o})]$, where W_{up} is the uplink bandwidth allocation per device and $\gamma_{\text{up},o}$ is the miner M_o's received signal-to-noise ratio (SNR). Following the same reasoning, global model downloading delay $T^{(\ell)}_{\text{dn},o}$ in Step 6 is given as $T^{(\ell)}_{\text{dn},o} = (h + \delta_{\text{m}} N_D) / [W_{\text{dn}} \log_2(1 + \gamma_{\text{dn},o})]$, where W_{dn} is the downlink bandwidth allocation per device and $\gamma_{\text{dn},o}$ is the device D_o's received SNR.

Communication delays are also incurred by Steps 3 and 5 among miners in the blockchain network. Assuming verification processing time is negligible compared to the communication delays, cross-verification delay $T^{(\ell)}_{\text{cross},o}$ in Step 3 is $T^{(\ell)}_{\text{cross},o} = \max\{T_{\text{wait}} - (T^{(\ell)}_{\text{local},o} + T^{(\ell)}_{\text{up},o}), \sum_{M_j \in \mathcal{M}\backslash M_o} \delta_{\text{m}} N_{M_j} / [W_{\text{m}} \log_2(1 + \gamma_{oj})]\}$ under frequency division multiple access (FDMA), where W_{m} is the bandwidth allocation per each miner link and γ_{oj} is the miner M_j's received SNR from the miner M_o. Similarly, denoting as $M_{\hat{o}} \in \mathcal{M}$ the miner who first finds nonce, referred to as the mining winner, total block propagation delay $T^{(\ell)}_{\text{bp},\hat{o}}$ in Step 5 is given as $T^{(\ell)}_{\text{bp},\hat{o}} = \max_{M_j \in \mathcal{M}\backslash M_{\hat{o}}} \{t^{(\ell)}_{\text{bp},j}\}$ under FDMA. The term $t^{(\ell)}_{\text{bp},j} = (h + \delta_{\text{m}} N_D) / [W_{\text{m}} \log_2(1 + \gamma_{\hat{o}j})]$ represents the block propagation delay from the mining winner $M_{\hat{o}}$ to $M_j \in \mathcal{M}\backslash M_{\hat{o}}$, and $\gamma_{\hat{o}j}$ is the miner M_j's received SNR from the miner $M_{\hat{o}}$.

Lastly, in Step 4, block generation delay $T^{(\ell)}_{\text{bg},j}$ of the miner $M_j \in \mathcal{M}$ follows an exponential distribution with mean $1/\lambda$, as modelled in [8]. The delay of interest is the mining winner $M_{\hat{o}}$'s block generation delay $T^{(\ell)}_{\text{bg},\hat{o}}$. Finally, the ℓ-th epoch latency $T_o^{(\ell)}$ is described as:

$$T_o^{(\ell)} = N^{(\ell)}_{\text{fork}} \left(T^{(\ell)}_{\text{local},o} + T^{(\ell)}_{\text{up},o} + T^{(\ell)}_{\text{cross},o} + T^{(\ell)}_{\text{bg},\hat{o}} + T^{(\ell)}_{\text{bp},\hat{o}} \right) + T^{(\ell)}_{\text{dn},o} + T^{(\ell)}_{\text{global},o}, \tag{4}$$

where $N^{(\ell)}_{\text{fork}}$ denotes the number of forking occurrences in the ℓ-th epoch, which follows a geometric distribution with mean $1/(1 - p^{(\ell)}_{\text{fork}})$, with the forking probability $p^{(\ell)}_{\text{fork}}$ at the ℓ-th epoch. Following Step 5, the forking probability is represented as:

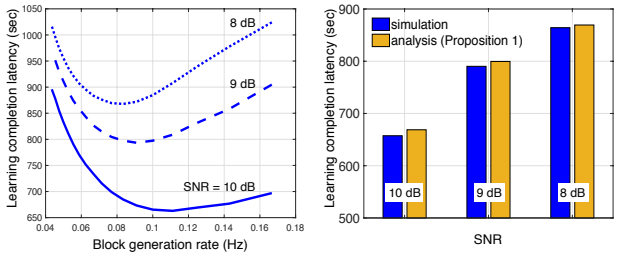
$$p^{(\ell)}_{\text{fork}} = 1 - \prod_{M_j \in \mathcal{M}\backslash M_{\hat{o}}} \Pr\left(t_j^{(\ell)} - t_{\hat{o}}^{(\ell)} > t^{(\ell)}_{\text{bp},j} \right), \tag{5}$$

where the term $t_j^{(\ell)} = T^{(\ell)}_{\text{local},j} + T^{(\ell)}_{\text{up},j} + T^{(\ell)}_{\text{cross},j} + T^{(\ell)}_{\text{bg},j}$ is the cumulated delay until the miner M_j generates a block.

B. Latency optimal block generation rate

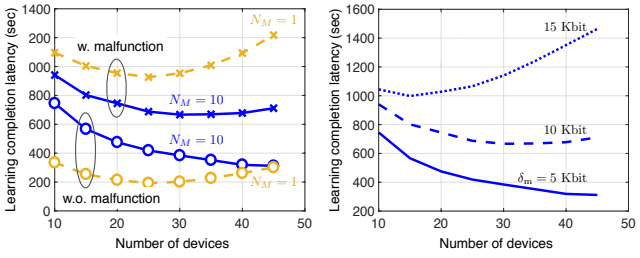
Using the one-epoch latency expression in (4), we aim at deriving the optimal block generation rate λ^* that minimizes the device D_o's ℓ-th epoch latency averaged over the PoW process. Here, the PoW process affects the block generation delay $T^{(\ell)}_{\text{bg},\hat{o}}$, block propagation delay $T^{(\ell)}_{\text{bp},\hat{o}}$, and the number $N^{(\ell)}_{\text{fork}}$ of forking occurrences, which are inter-dependent due to the mining winner $M_{\hat{o}}$. Solving this requires to compare the cumulated delays for all miners and their associated devices under their asynchronous operations that complicate the optimization.

In order to avoid the said difficulty, we consider the case where all miners synchronously start their PoW processes by adjusting T_{wait} such that $T^{(\ell)}_{\text{cross},o} = T_{\text{wait}} - (T^{(\ell)}_{\text{local},o} + T^{(\ell)}_{\text{up},o})$. In this case, even the miners completing the cross-verification earlier wait until T_{wait}, thus providing the performance lower bound, i.e., latency upper bound. With this synchronous approximation to the exact operations, we derive the optimal block generation rate λ^* in a closed form, as provided next.



(a) With respect to λ. (b) With λ^*.

Fig. 3. Average learning completion latency (a) versus block generation rate λ and (b) with the optimum λ^* ($\gamma_{\mathrm{up},o} = \gamma_{\mathrm{dn},o} = \gamma_{oj} = \mathsf{SNR}$).



(a) With/without malfunction. (b) For different model sizes.

Fig. 4. Average learning completion latency versus the number of devices, (a) under the miners' malfunction and (b) for different local model sizes.

Proposition 1. With the PoW synchronous approximation, i.e., $T^{(\ell)}_{\mathrm{cross},o} = T_{\mathrm{wait}} - (T^{(\ell)}_{\mathrm{local},o} + T^{(\ell)}_{\mathrm{up},o})$, the block generation rate λ^* minimizing the ℓ-th epoch latency $\mathbb{E}[T^{(\ell)}_o]$ averaged over the PoW process is given by:

$$\lambda^* \approx 2\left(T^{(\ell)}_{\mathrm{bp},\hat{o}}\left[1+\sqrt{1+4N_M\left(1+T_{\mathrm{wait}}/T^{(\ell)}_{\mathrm{bp},\hat{o}}\right)}\right]\right)^{-1}.$$

Proof: Applying the synchronous PoW approximation and the mean $1/(1-p^{(\ell)}_{\mathrm{fork}})$ of the geometrically distributed $N^{(\ell)}_{\mathrm{fork}}$ to (4),

$$\mathsf{E}[T^{(\ell)}_o] \approx \left(T_{\mathrm{wait}} + \mathsf{E}[T^{(\ell)}_{\mathrm{bg},\hat{o}}]\right) / \left(1-p^{(\ell)}_{\mathrm{fork}}\right) + T^{(\ell)}_{\mathrm{dn},o} + T^{(\ell)}_{\mathrm{global},o}. \quad (6)$$

The terms T_{wait}, $T^{(\ell)}_{\mathrm{dn},o}$, $T^{(\ell)}_{\mathrm{global},o}$ are constant delays given in Sect. II-A. The remainder is derived as follows.

For the probability $p^{(\ell)}_{\mathrm{fork}}$, using (5) with $t^{(\ell)}_j - t^{(\ell)}_{\hat{o}} = T^{(\ell)}_{\mathrm{bg},j} - T^{(\ell)}_{\mathrm{bg},\hat{o}}$ under the synchronous approximation, we obtain $p^{(\ell)}_{\mathrm{fork}}$ as:

$$p^{(\ell)}_{\mathrm{fork}} = 1 - e^{\lambda \sum_{M_j \in \mathcal{M}\setminus M_{\hat{o}}} T^{(\ell)}_{\mathrm{bp},j}}, \quad (7)$$

where $T^{(\ell)}_{\mathrm{bp},j}$ is a constant delay given in Sect. II-A. Next, for the delay $\mathsf{E}[T^{(\ell)}_{\mathrm{bg},\hat{o}}]$, using the definition of $T^{(\ell)}_{\mathrm{bg},\hat{o}}$ and the complementary cumulative distribution function (CCDF) of the exponentially distributed $T^{(\ell)}_{\mathrm{bg},j}$, we derive $T^{(\ell)}_{\mathrm{bg},\hat{o}}$'s CCDF as:

$$\Pr\left(T^{(\ell)}_{\mathrm{bg},\hat{o}} > x\right) = \prod_{j=1}^{N_M} \Pr\left(T^{(\ell)}_{\mathrm{bg},j} > x\right) = e^{-\lambda N_M x}. \quad (8)$$

Applying the total probability theorem yields $\mathsf{E}[T^{(\ell)}_{\mathrm{bg},\hat{o}}] = 1/(\lambda N_M)$. Finally, combining all these terms, (6) is recast as:

$$\mathsf{E}[T^{(\ell)}_o] \approx (T_{\mathrm{wait}} + 1/\lambda N_M)\, e^{\lambda \sum_{M_j \in \mathcal{M}\setminus M_{\hat{o}}} T^{(\ell)}_{\mathrm{bp},j}} + T^{(\ell)}_{\mathrm{dn},o} + T^{(\ell)}_{\mathrm{global},o}, \quad (9)$$

which is convex with respect to λ. The optimum λ^* is thus derived from the first order necessary condition. ■

The accuracy of the above result with the synchronous approximation is validated by comparing the simulated λ^* without the approximation in the following section.

IV. Numerical Results and Discussion

In this section, we numerically evaluate the proposed blockFL's average learning completion latency $\mathsf{E}[T_o] = \sum_{\ell=1}^{L} \mathsf{E}[T^{(\ell)}_o]$. By default, we consider $N_D = N_M = 10$, and $N_i \sim \mathrm{Uni}(10, 50)$ $\forall D_i \in \mathcal{D}$. Following the 3GPP LTE Cat. M1 specification [15], we use $W_{\mathrm{up}} = W_{\mathrm{dn}} = W_{\mathrm{m}} = 300$ KHz and $\gamma_{\mathrm{up},o} = \gamma_{\mathrm{dn},o} = \gamma_{oj} = 10$ dB. Other simulation parameters are given as: $\delta_{\mathrm{d}} = 100$ Kbit, $\delta_{\mathrm{m}} = 5$ Kbit, $h = 200$ Kbit, $f_c = 1$ GHz, and $T_{\mathrm{wait}} = 50$ ms.

Fig. 3 shows the impact of block generation rate λ on the BlockFL's average learning completion latency. In Fig. 3-a, we observe that the latency is convex-shaped over λ and is decreasing with the SNRs. In Fig. 3-b, for the optimal block generation rate λ^*, the minimized average learning completion latency time obtained from Proposition 1 is always longer by up to 1.5% than the simulated minimum latency without the approximation, as predicted in Sect. III-B.

Next, Fig. 4 illustrates the BlockFL's scalability in terms of the numbers N_M and N_D of miners and devices, respectively. In Fig. 4-a, the average learning completion latency is computed for $N_M = 1$ and $N_M = 10$ with or without the miners' malfunction. The malfunction is captured by adding Gaussian noise $\mathcal{N}(-0.1, 0.01)$ to each miner's aggregate local model updates with probability 0.5. Without any malfunction, a larger N_M increases the latency due to the increase in their cross-verification and block propagation delays. This does not always holds under the miners' malfunction. In BlockFL, each miner's malfunction only distorts its associated device's global model update. Such distortion can be restored by federating with other devices that associate with the miners operating normally. For this reason, a larger N_M may achieve a shorter latency, as observed for $N_M = 10$ with the malfunction.

Lastly, Figs. 4-a and b describe that there exists a latency-optimal number N_D of devices. In fact, a larger N_D may decrease the learning completion latency thanks to utilizing a larger amount of data samples. Meanwhile, on the contrary, it increases each block size, i.e., communication payload, and thus leads to higher block exchange delays, consequently resulting in the convex-shaped latency with respect to N_D. In this respect, a proper device selection has a potential to reduce the latency, as investigated in [5], [10]. Finally, Fig. 4-b shows that the latency increases with each device's local model size δ_{m}. Thus, it calls for a model compression technique, which could be an interesting topic for future research.

References

[1] P. Popovski, J. J. Nielsen, C. Stefanovic, E. de Carvalho, E. G. Ström, K. F. Trillingsgaard, A. Bana, D. Kim, R. Kotaba, J. Park, and R. B. Sørensen, "Wireless Access for Ultra-Reliable Low-Latency Communication (URLLC): Principles and Building Blocks," *IEEE Netw.*, vol. 32, pp. 16–23, Mar. 2018.

[2] M. Bennis, M. Debbah, and V. Poor, "Ultra-Reliable and Low-Latency Wireless Communication: Tail, Risk and Scale," [Online]. Available: https://arxiv.org/abs/1801.01270.

[3] J. Park, D. Kim, P. Popovski, and S.-L. Kim, "Revisiting Frequency Reuse towards Supporting Ultra-Reliable Ubiquitous-Rate Communication," *in Proc. IEEE WiOpt Wksp. SpaSWiN, Paris, France*, May 2017.

[4] J. Konečný, H. B. McMagan, D. Ramage, "Federated Optimization: Distributed Machine Learning for On-Device Intelligence," [Online]. Available: https://arxiv.org/abs/1610.02527.

[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A.y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *in Proc. AISTATS, Fort Lauderdale, FL, USA*, Apr. 2017.

[6] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting Distributed Synchronous SGD," *in Proc. ICLR, San Juan, Puerto Rico*, May 2016.

[7] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[8] C. Decker, and R. Wattenhofer, "Information Propagation in the Bitcoin Network," *in Proc. IEEE P2P, Trento, Italy*, Sep. 2013.

[9] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed Federated Learning for Ultra-Reliable Low-Latency Vehicular Communications," [Online]. Available: https://arxiv.org/abs/1807.08127.

[10] T. Nishio, R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," [Online]. Available: https://arxiv.org/abs/1804.08333., 2018.

[11] R. Johnson and T. Zhang, "Accelerating Stochastic Gradient Descent Using Predictive Variance Reduction," *in Proc. NIPS, Lake Tahoe, NV, USA*, Dec. 2013.

[12] O. Shamir, N. Srebro, and T. Zhang, "Communication-Efficient Distributed Optimization Using An Approximate Newton-Type Method," *in Proc. ICML, Beijing, China*, Jun. 2014.

[13] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On Security Analysis of Proof-of-Elapsed-Time," *in Proc. SSS, Boston, MA, USA*, Nov. 2017.

[14] V. Costanand, and S. Devadas, "Intel SGX Explained," *Cryptologye Print Archive Report 2016/086*, 2016.

[15] 3GPP TS 36.300 v13.4.0, "E-UTRA and E-UTRAN; Overall Description; Stage 2," Tech. Rep., 2016.